# Java Multimedia Telecollaboration

J. C. de Oliveira[1,2], M. Hosseini[1], S. Shirmohammadi[1], F. Malric[1], S. Nourian[1], A. El Saddik[1] & N. D. Georganas[1]

[1] Multimedia Communications Research. Laboratory
School of Information Technology and Engineering
University of Ottawa
161 Louis Pasteur Priv.,
Ottawa, ON K1N 6N5
CANADA
+1 (613) 562-5800x6248

[2] Departament of Computer Science
National Laboratory for Scientific Computation
Av. Getúlio Vargas, 333
Petrópolis, RJ 25651-070
BRAZIL
+55 (24) 2233-6022

jauvane@acm.org, [mojtaba,shervin,frank,saeid,abed,georganas]@mcrlab.uottawa.ca

## ABSTRACT

This paper discusses different design possibilities of multimedia collaborative environments. The main function of such environments is to share multimedia resources among several geographically distributed users. To optimize the use of bandwidth and compensate for latency, we have chosen an approach that sends as small amount of information as possible for the updates, namely events. Therefore, we will describe six different architectures to achieve such a goal. These prototypes (jStreaming, JETS, JASMINE, JASBER and two Collaborative Virtual Environments) are fully written in Java.

## Keywords

Java, Collaboration, CSCW, Multimedia, Video Streaming, H263, jStreaming, JETS, JASBER, JASMINE, Shared Applets, MPEG4.

## 1. INTRODUCTION

Multimedia Systems have long been used for Collaboration. In order to make better use of the web for collaboration amongst a group of individuals, such systems had to be implemented for various platforms. Lawrence Berkeley National Laboratory/UC Berkeley's VIC [5] is a good example of such an effort, as there are several versions compiled for various flavours of Unix (including Linux), Windows, etc. It would be beneficial to be able to have a single build for the various platforms so that updates would need to be made only to that code. Java [12] shows a great potential in that respect since it provides a "Compile Once-Run Everywhere" architecture. A comprehensive number of Operating Systems support Java, allowing any one of those systems to benefit from Java compliant applications. Java Applets brings extra features such as the ability to run an Applet within a web-browser with the executable code downloaded at run-time from a (web) server. This allows making updates in the code in that single location (the server) which automatically ensures that every client will be running the exact same version of the code. No other language allows such easy maintenance. A developer has no longer to worry about making a new system compatible with all previous versions in order to welcome users with non up-to-date versions of the software.

It has been shown [1, 2, 3] that Java has a great potential for Collaborative Multimedia Systems with acceptable performance in rather complicated prototypes.

If a developer adheres to a subset of Java, known as PersonalJava, one may ensure an even greater client base, as any Java Enabled operating system is also PersonalJava enabled. Many OS which cannot handle the complete Java set have some PersonalJava implementation available. One such example is the OS for the PDA market, including Windows CE on the PocketPCs.

We have developed, at the Multimedia Communications Research Laboratory at the University of Ottawa, a comprehensive set of Java and PersonalJava compliant Multimedia Systems aiming at Collaboration amongst a group of users.

In section 2, we will introduce jStreaming, a 100% Pure Java™ Video Decoder for H.263 video streams. Section 3 introduces JETS, a 100% Pure Java™ Java Enabled Telecollaboration System. Section 4 presents JASMINE, which is a Java System allowing users to transparently share any Applet. Section 5 introduces JASBER, a prototype that allows a group of users to collaboratively browse the web. Section 6 outlines the utilization of Java in developing Distributed Virtual Environments. Section 7 presents related work followed by the conclusion.

## 2. jSTREAMING

jStreaming [13] is a video decoder of standard ITU-T H.263 [4] video streams fully written in Java. Being Java compliant allows jStreaming to run in every single Java enabled web browser when running as an Applet as well as any Java Enabled OS when running as a Java Application. Figure 1 shows an Applet version of jStreaming with a standard video stream being played.

**Figure 1. jStreaming as an Applet**

jStreaming's core complies with JDK 1.0.2 (the first release of Java broadly available for the public) and, as such, is also compliant with more recent implementations of Java (JDK 1.1.x, 1.2, 1.3, etc). Such compatibility allows jStreaming to be deployed even by older versions of web browsers, such as Netscape Navigator 3.0 and Microsoft Internet Explorer 3.02, etc. Additionally, jStreaming also complies with PersonalJava, a subset of JDK 1.1 which addresses more limited devices such as PocketPCs and other PDAs. JStreaming hence runs smoothly on such devices. We have achieved a 10fps playback rate, QCIF, with a Compaq iPAQ 3670 PocketPC and we have received reports of jStreaming achieving over 144fps under Windows. Figure 2 shows jStreaming running in an iPAQ 3650 PocketPC (Windows CE 3.0). In a PII 333MHz we achieved 34fps back in 1998. Native code achieves about 55-60fps in the same system.

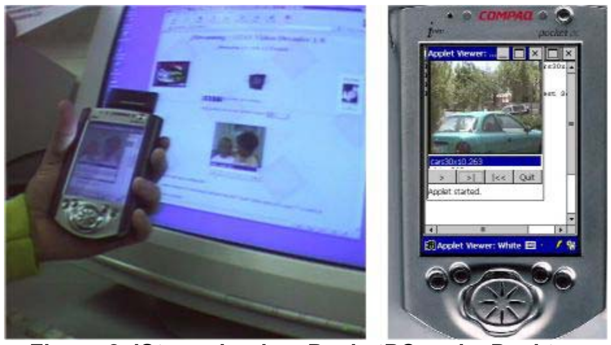


**Figure 2. jStreaming in a PocketPC and a Desktop**

jStreaming provides a high-level API, which allows it to be bundled into other prototypes (See section 3 on JETS for an example).

jStreaming can stream video from a Multithreaded VideoServer (also written in Java) through a simple protocol with sliding window flow control as well as from a web server through HTTP.

Figure 3 shows the architecture of jStreaming for live streaming, using a native code encoder. jStreaming itself is the technology starting at the Video Server box all the way to the client Applet/Application.

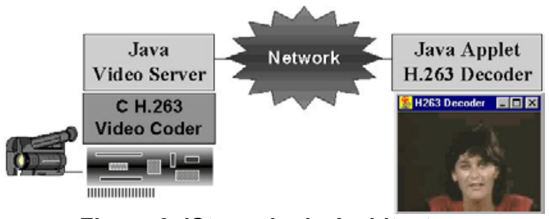


**Figure 3. jStreaming's Architecture**

## 3 JETS 2000

JETS (Java Enabled Telecollaboration System) [7, 8, 9, 14] is a client-server framework that permits sharing of Java applets and applications. Since JETS uses the core Java packages, users don't need to install any additional Java classes on their system. This allows any user to access JETS and share applets with a Java-enabled browser. JETS 2000, the latest version of JETS, also offers video-conferencing using the Java Media Framework (JMF). Figure 4 below shows a screenshot of a sample JETS session.

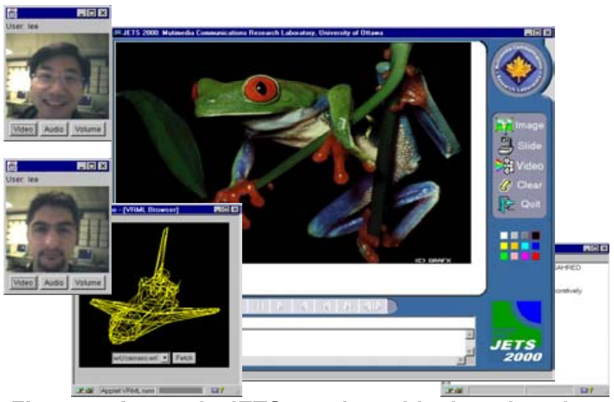


**Figure 4. A sample JETS session with shared applets and A/V conferencing.**

As can be seen from the figure 4, JETS consists of many utilities that enable multimedia viewing and sharing.

### 3.1. Whiteboard

The whiteboard is an interactive space where clients on a virtual session can share ideas such as pictures, slides, text, video or drawings. The users can annotate on these images and start a discussion. The built-in locking mechanism of JETS is used to forbid modification of the same object at the same time by more than one user. Figure 5 shows JETS' main interface consisting of the control panel (right), a chatting dialog box (lower left) end and the shared whiteboard area (top-left).

The simplest way of interacting on the whiteboard is through text. Clients can use the chat area to communicate. Every other member who has input access to the whiteboard will see the originator of the message followed by the message itself.

Another way of interacting through the whiteboard is by drawing. To do this, the user simply chooses a color from the color template and draws by keeping the left mouse button down and dragging the mouse around the screen. A user can clear all the annotations (drawings) by pressing the "Clear" button.

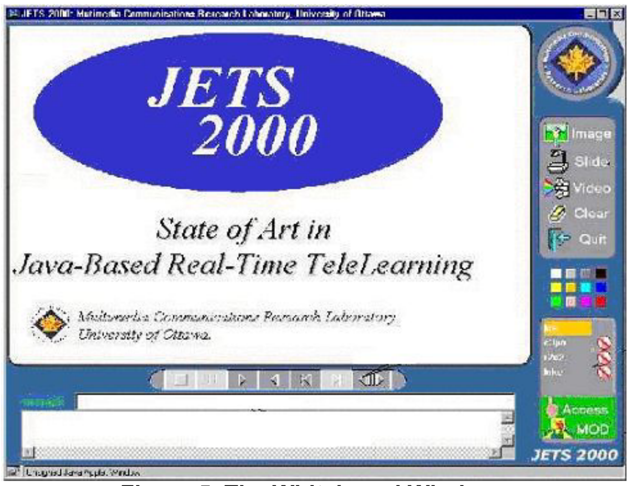

JETS 2000: Multimedia Communications Research Laboratory, University of Ottawa
JETS 2000
State of Art in Java-Based Real-Time TeleLearning
Multimedia Communications Research Laboratory University of Ottawa
Image
Slide
Video
Clear
Quit
Access MOD
JETS 2000

**Figure 5. The Whiteboard Window**

A client may paste a picture found in archive by pressing the "Image" button to the right and choosing one file in the image file dialog box. The picture will instantly appear to all other members' whiteboard. Any member having full access can freely comment or draw onto the picture.

A client may also start a slide show found in archive by pressing the "Slide" button to the right and choosing one file in the Slide Show dialog box. Any member having full access can freely comment or draw onto the slide show as well as go to the next or previous slide using the appropriate VCR buttons respectively.

## 3.2. Shared H.263 Video Presentation

A very useful feature of JETS is the ability to play ITU-T H.263 compliant video in the whiteboard. That is accomplished using jStreaming's API. When a user opens a video file and starts playing it, the video data is streamed down to all participants, decoded in real-time (processor permitting) and displayed in their whiteboard. Figure 6 shows a video being displayed with some annotation drawn on top of it.

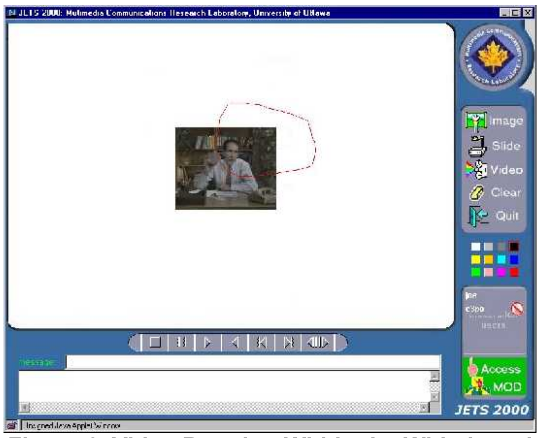


**Figure 6. Video Running Within the Whiteboard**

## 3.3. VRML Viewer

Another sample applet is a simple shared 3D viewer for VRML files which permits real-time collaborative interaction with simple VRML objects. The applet brings from the server simple VRML 1.0 files and displays them in wire frame mode. A user can then collaboratively interact with the 3D object, with all the rotations, translations and zooming reflected on all participants' screens. Figure 7 shows the shared VRML browsing interface.

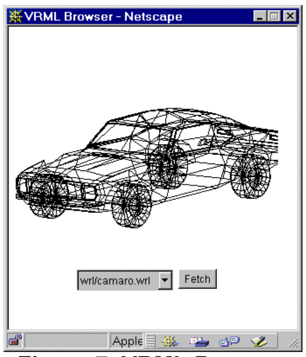


**Figure 7. VRML Browser**

## 3.4 Video Conferencing and Recording with J-VCR

The *Java Video Conference Recorder (J-VCR)* tool further enhances JETS 2000 by providing services for audio/video conferencing, recording a session, and playback of a recorded session. J-VCR can record the session in the Synchronous Multimedia Integration Language (SMIL) format, which is a World Wide Web Consortium (W3C) standard. As a result, any SMIL-player such as RealNetwork's RealPlayer can be used to playback the recorded session [11].

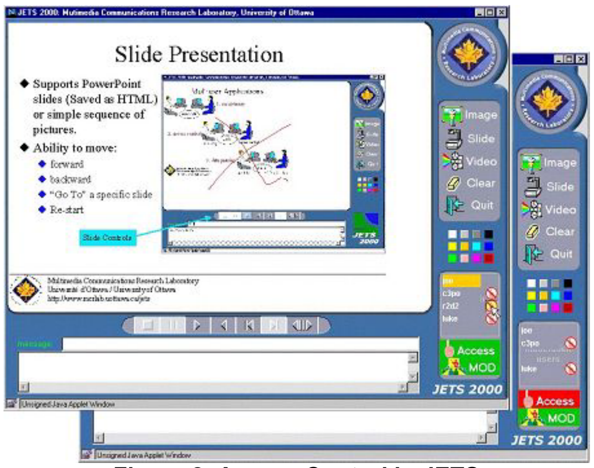


**Figure 8. Access Control in JETS**

## 3.5. Session Management

JETS implements a management system which enables monitoring of the session. One of the session clients has to log in as a moderator by clicking the "MOD" button to start a session management. Once the session management has been established, only the moderator has the right to access the shared whiteboard and other clients have no access. Every session client can ask the moderator for access permission by simply clicking his/her "Access" button. The moderator might either grant or reject that client's request. Once the moderator approves a session client's request, this client therefore become a session participant and gains the right to access the shared whiteboard. Any session participant can put his/her notation on the shared whiteboard. The moderator has the right to revoke access privileges to any client at any time but allowing the refused client to ask the moderator for permission again if desired. Figure 8 shows two JETS windows. The one in the foreground with a green "Access" icon indicates that such client is allowed to interact with the whiteboard. On the other hand the one in the background with a red "Access" button indicates that such user can only see what is being done in the whiteboard (not being able to interact any further with it).

## 3.6 Architecture

From a developer's point of view, JETS can be regarded as a set of Application Programming Interfaces (API) that the developer can use to build shared resources. It provides the developer with built-in consistency, access control, and data passing.

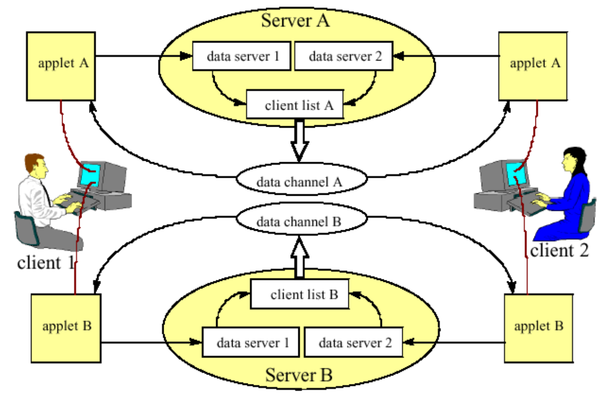


**Figure 9. Client-Server communication in JETS**

JETS uses a multithreaded server as shown in figure 9, where the main server launches a sub-server for each user joining the session. The sub-server is responsible for processing only the update messages or requests coming in from its own client. Once the sub-server receives the update message, it will send it to all other clients in the session. This will create a fast system response, at the expense of more resources utilized due to sub-server threads. However, usually only one client at a time can control and interact with an application (due to floor control in session management), and most threads will simply be waiting. For its client-server communication, JETS uses TCP/IP and UDP/IP sockets. In figure 9, when client 1 does some interaction with application A, his actions are reflected to data server 1 which runs as part of Server A for application A. Next, data server 1 relays the actions of client 1 to other clients which are listed in a client list on Server A. Finally, application A of client 2 receives those actions and reflects them on the screen of client 2.

## 3.7 Performance Evaluation

JETS can be considered a real-time tool in the sense that its updating response time, in a network environment capable of supporting real-time applications, is within the acceptable parameters of human quality of service for desktop collaboration, as we shall see. But as with any TCP based multiuser system, there is an upper-bound to the number of simultaneous users before those parameters are violated. This "maximum users" limit depends on the resources utilized by the system, such as processing power, graphics power, memory, network bandwidth and network delay, as well as the design of the communication part of the system.

Depending on the quality desired, the application level end-to-end delay between two users should be less than 1000 milliseconds, with 200 milliseconds recommended for tightly-synchronized tasks [11]. However, these numbers are valid only if the shared application is used in conjunction with some type of media that provide a sense of presence such as video and audio. The reason is that if audio or video or both are present, users have a sense of "awareness" of each other, which in turn requires the shared application to respond within a time that maintains that awareness. For example, imagine three engineers who are collaboratively designing a bridge in a live session. One of them highlights a section of the bridge and says: "I think this part should be redesigned". If they are using real-time audio conferencing (end-to-end audio delay of 100 msec), then the delay of the shared application must comply with the above numbers in order for the other two engineers to receive the audio message and the event update in such a way as to maintain the real-time quality of the session. This is usually the case in controlled IP environments such as local networks or corporate IP networks.

In the case of typical Internet connections, where audio and video delays are not controllable, or in the absence of audio or video, restrict delay parameters make little sense because the users have no time-wise perception of one another. In such instances, when a user receives an update message, the user has no way of knowing when an actual action occurred. So, even a delay of 5 seconds or more might be acceptable depending on the nature of the application under such circumstances.

Our performance evaluations are done for a controllable environment, where real-time characteristics are required and can be supported.

### 3.7.1 Parameters of Interest

The most common parameter that measures the quality of a collaborative application is the Client-to-Client Delay (CCD). CCD tries to measure the average time it takes for an update message to reach other users. It includes all layers between the two clients, including application, transport, networking, and physical layer delays. However, at the application level, it only measures the time it takes for a sender to send or a receiver to receive the update at the application layer. It does not include the delay caused by what the application does with the update because that is application-dependent. As an example, if one user opens an image in a whiteboard, what we measure is how long it takes for the "open-image" message to reach all clients. We don't measure how long it takes for the receivers to actually download the image from the given URL and show it on their screen, because we can't control those delays with JETS server.

In addition, the server processing time per packet increases with increasing number of simultaneous users. This is due to one-to-one TCP connection-oriented nature of the system; the server needs to send the update info to each client one by one. This Server Processing Delay (SPD) adds to the overall end-to-end delay of the system and must be taken into account when calculating maximum number of users supported by the system. Another interesting parameter is the Floor Control Delay (FCD). This is the average time for a user to take control or be denied taking control of an application and measures how intuitive a system is. A system with a smaller FCD is more "natural" and behaves more naturally than a system with a larger FCD.

### 3.7.2 Testing and Results

We tested CCD, SPD, and FCD of JETS over both 100Mbps local area network (LAN) and 28.8Kbps telephone modem access. During the testing, all machines were running their usual background processes related to the network and the operating system. [6].

### 3.7.3 CCD Test

For the CCD test, we had a "sender" applet send an event to a "receiver" applet. Upon receiving the event, the receiver applet extracts all necessary data from the packet, reassembles the event, and sends the event back to the sender. The sender does the same thing and resends the event, and so on. This is repeated for a given duration, which was 10 minutes in our tests. The result is shown is figure 10. The packet size is measured in number of integers sent per packet. Typical packet sizes can be from 3 integers (draw a point with given colors) to 8 (draw line from point A to B with given color) to larger sizes. Even though it is very unlikely that a message of size 300 integers is sent in one packet, we did extend our test to that limit to see the effect of very large update messages. Figure 11 shows the same test performed over 28.8 Kbps modem access instead of 100 Mbps Ethernet.

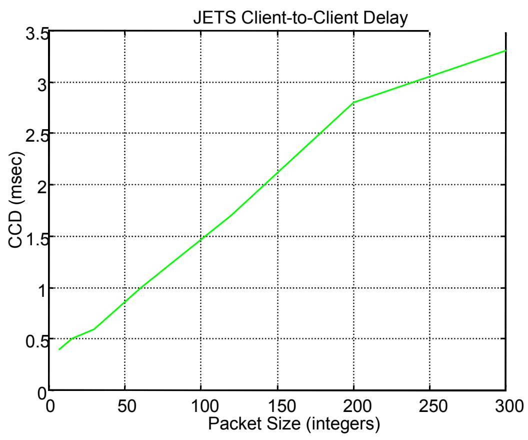


**Figure 10. JETS CCE results (packet-based)**

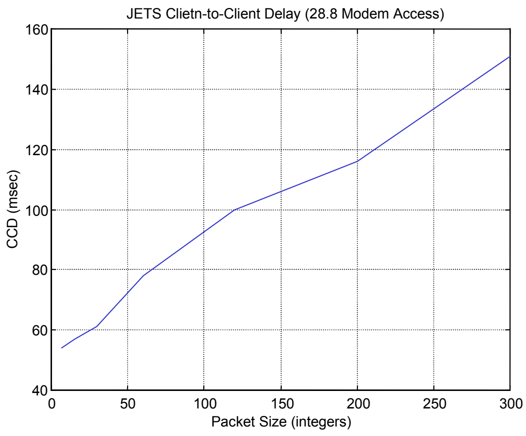


**Figure 11. JETS CCD results over a modem line (packet-based).**

**3.7.4 SPD Test**

For the SPD test, we had the sender applet flood the server with event updates. Then we had the receivers (up to 45) calculate the average delay between receiving adjacent packets from the server. As expected, this delay increases with increasing number of users as seen in figure 12 for an update message of size 8 integers. Figure 13 shows the same test performed for updates of various sizes. Note that due to floor control and moderation, no more that one client at a time can send events to the server, a scenario, which is typical of collaborative applications.

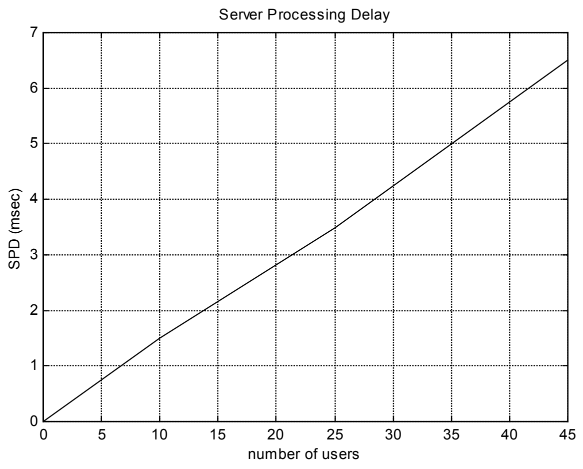


**Figure 12. JETS Server Processing Delay**

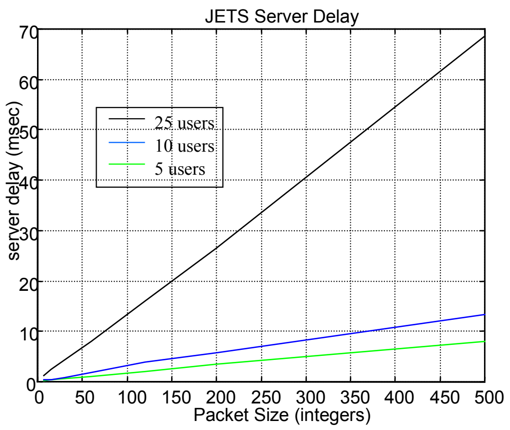


**Figure 13. JETS Server Processing Delay (data)**

We can see that the delay increases linearly. This is due to the fact that the server spends an equal amount of processing time per packet per client; therefore it increases linearly with increasing number of clients.

**3.7.5 FCD Test**

For the Floor Control Delay, we had a client constantly ask for control, and release it upon receipt, for a given amount of time. The average FCD turned out to be less than 5 msec, which affirms the intuitiveness of the floor-control mechanism of the system.

**3.7.6 Analysis**

As mentioned before, the recommended overall end-to-end delay is less than 1000 msec, with less than 200 msec required for closely-coupled collaboration. This delay includes the CCD, the SPD, and the on-screen rendering/display delay corresponding to the application's GUI. The rendering delay (RD) is not constant and it depends on the hardware/OS/platform used.

From the CCD and SPD tests, we can approximate the overall delay as:

delay = CCD + SPD + RD;

from Figure 12: SPD ≈ 0.142* N, where N is the number of users; hence:

$$\boxed{\text{delay} \approx \text{CCD} + 0.142^* \text{ N} + \text{RD}}$$

which roughly represents the delay experienced from the time a typical event is generated due to a client's interaction until that interaction is shown on the screen of all other clients. Figure 14 shows the achievable number of users based on the expected overall delay, for different rendering delays (RD).

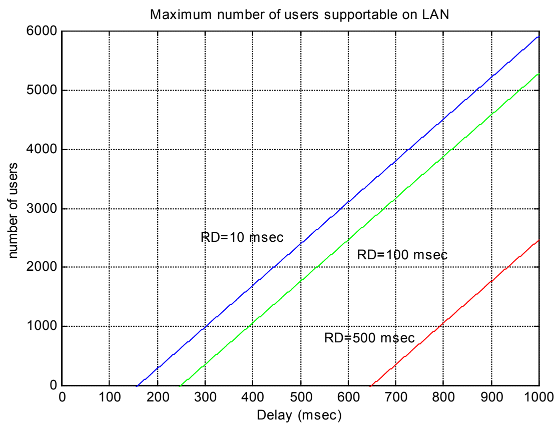


**Figure 14. Number of users supported by the system**

Figures 15 shows the same thing with focus on tightly-synchronized tasks (delay<200 msec).

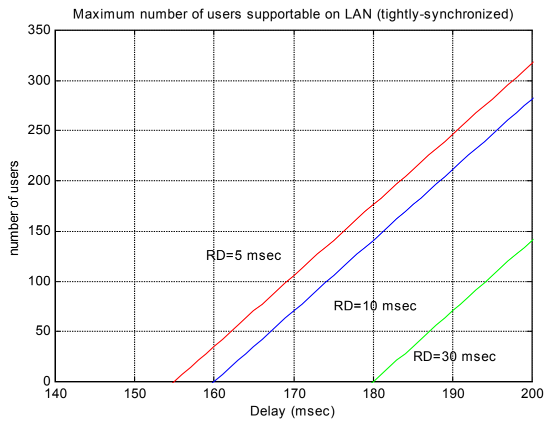


**Figure 15. Number of Users supported by the system (delay < 200 msec)**

Finally, figure 16 illustrates the number of users supportable with 28.8 Kbps modem access.

By looking at the above graphs, we can conclude that the system can support "many" users. Even though the plots suggest that theoretically thousands of users can be supported, the fact is that the actual number of users supportable is less. The reason is that the linear behavior of the system diminishes as the number of users increases: the performance of the machine(s) running the server decreases substantially as we approach the limit of maximum allowable socket connections on the equipment; also the underlying physical network becomes slower with increasing number of users. So the hardware/OS of the server machine and the network either cannot support so many simultaneous users, or their performance decreases significantly. Nevertheless, this shows that the underlying communication module of JETS can support small-size and medium-sized collaboration sessions of hundreds of users, resource permitting.

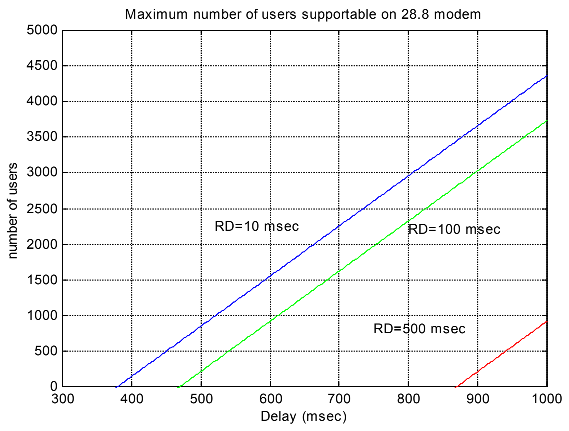


**Figure 16. Number of users supported by the system (modem access)**

## 4 JASMINE

JASMINE (Java Application Sharing in Multi-user INteractive Environments) [6] is a prototype which allows a group of users to share almost any Java applet or application available on the Internet. The idea is that any applet not designed to be used collaboratively can be used as such through JASMINE.

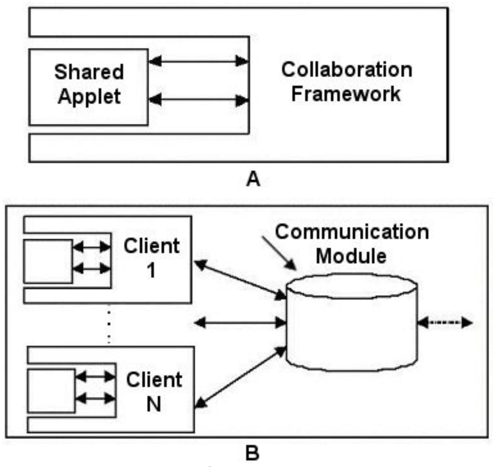


**Figure 17. JASMINE's Architecture**

JASMINE takes advantage of the many useful resources and objects that have been developed as Java applets or applications and which are available on the Internet. One can access these resources by simply downloading them from different repositories. In many instances, a group of users may wish to share these resources in real-time, such as when an instructor teaches remote students how to use a certain resource or explains to them the theory behind it. JASMINE allows the sharing of these Java applets and applications in real-time without requiring modification of their code as well as allowing an instructor to dynamically manage the collaborative session as detailed below.

Figure 17 illustrates the overall concept with JASMINE framework wrapping around an applet that is to be shared (Figure 17.a). The framework listens to all events occurring in the graphical user interface of the applet and transmits these events to all other participants in order to be reconstructed there. The framework captures both Java AWT-based and Swing-based events. After capturing the event, it is sent to the communication module where the event is sent to all other participants in the session (Figure 17.b).

Figure 18 shows a sample JASMINE session, where arbitrary applets and resources from the Internet have been brought into the session dynamically. Since any Java applet or application can be brought into the session, JASIMNE has an unlimited extendibility with each resource enhancing JASMINE dynamically.

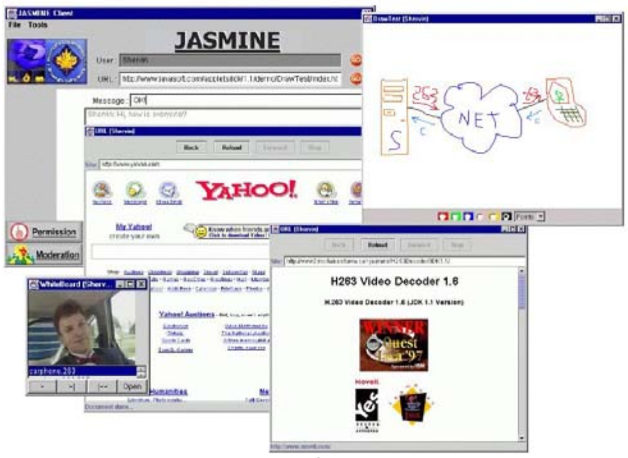


**Figure 18. Main JASMINE Application**

JASMINE also allows one to manage the session and granting access to specific parties. Such a feature is useful for environments where some tighter control is needed such as teletraining where the lecturer may wish to prevent students from changing the presentation without permission similar to the real world moderation of a classroom or meeting. Figure 19 below shows such features of JASMINE

As shown in Figure 19.a, an instructor who has logged into the session with a special password can enable moderation by pressing the moderation button. Once this button is pressed, any participant wishing to interact with the shared applications must ask for permission by pressing the permission button. The moderator receives each user's requests which can each be granted or denied (Figure 19.b). Should the moderator decide to grant permission, the participant sees a green light on the permission button (Figure 19.c) and can interact with the application. The moderator can "cut off" any participant by pressing the cut button next to the participant's name (Figure 19.d).

The architecture was developed aiming at enabling collaboration via collaborative-unaware applications and applets without the need for modifying the source code of such applets. JASMINE's architecture enables the use of almost all single-user applets and applications in a collaborative way. With the popularity and widespread use of computing environments where Java applications and applets are running over IP, JASMINE's architecture helps people to collaborate in such environments easily.

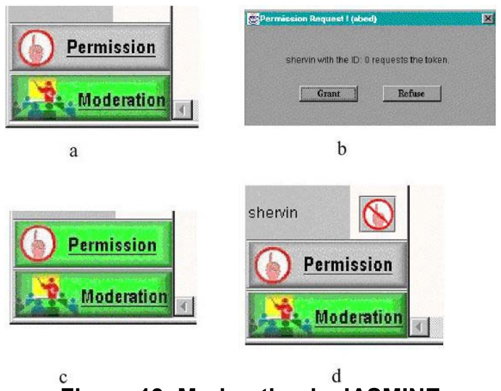


**Figure 19. Moderation in JASMINE**

The JASMINE client is responsible for capturing events, sending events to the server, receiving events from the server and reconstructing events locally. It is a Java application that consists of four important components:

- Collaboration Manager
- Listener Adapter
- Component Adapter
- Event Adapter

The **Collaboration Manager** is the main component of the client side. It is responsible for communication between clients and the server and it provides GUI as well. The **Listener Adapter** implements several AWT listeners. After catching an event, the Listener Adapter converts the event to the remote event and forwards it to the Collaboration Manager. The **Component Adapter** maintains a list of references to the GUI components of all applications and applets. This list is created in the same order on each client, so each component has the same reference number on all clients. The **Event Adapter** works opposite to the Listener Adapter. It converts the remote events to local events and applies them to corresponding components.

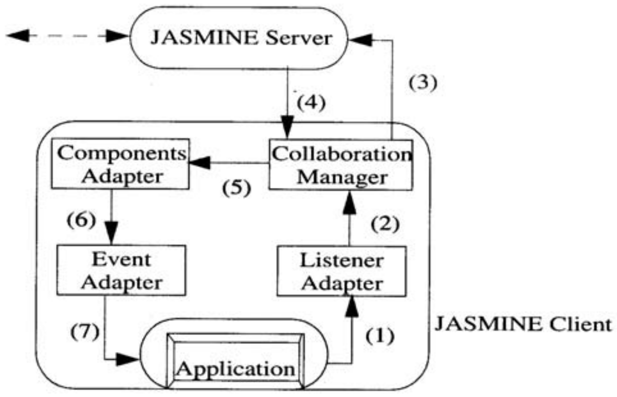


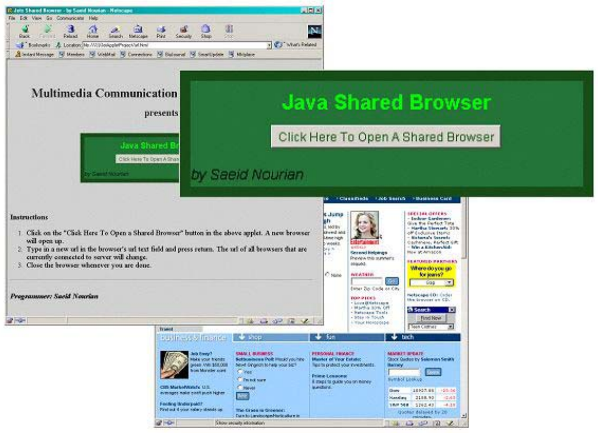
**Figure 20. JASMINE data flow diagram**

The Figure 20 presents the date flow diagram of the client side architecture. There are two main data paths in the system: the first path is labeled with 1, 2, and 3. Any Java event occurring in a Java application is caught by the Listener Adapter. If the event is a local event, the Listener Adapter converts it to a remote event and forwards it to the Collaboration Manager. Then the Collaboration Manager sends the remote event to the JASMINE server. The second path is label with 4, 5, 6, and 7. The remote event received by the Collaboration Manager is forwarded to the Component Adapter. The Component Adapter gets the information about the source of the event and sends it with the event to the Event Adapter. The Event Adapter converts the remote event to a normal AWT event and dispatches the event to the corresponding component.

Notice that the JASMINE server uses the same communication techniques as the JETS server and therefore has the same type of performance characteristics as JETS in terms of network delay and other parameters discussed in 3.7.

## 5 JASBER

JASBER (JAva Shared BrowsER) allows a group of users to share their Netscape browser. In other words, any change in the HTML content of a given shared browser will be reflected on all other shared browsers. Figure 21 shows JASBER's interface. JASBER is a client applet that communicates with a JETS server. Once the applet is loaded, a client/server connection is established. The user can then open a shared browser by clicking on the appropriated button. Any change made in the HTML content of that browser (i.e. changing the URL) will result in an update in the HTML content of the other shared browsers that are communicating with the server.



Instructions

1. Click on the "Click Here To Open a Shared Browser" button in the above applet. A new browser will open up.
2. Type in a new url in the browser's url text field and press return. The url of all browsers that are currently connected to server will change.
3. Close the browser whenever you are done.

*Programmer: Saeid Nourian*

**Figure 21. JASBER's Interface**

JASBER ensures consistency amongst the various clients through the following procedure:

- Whenever a user loads a new web page by typing a new web site address (URL) in the address field of the shared browser, JASBER will detect such action and forward the new address to the server. The server then forwards it to all JASBER clients currently in the session. As a result all shared browsers that are connected to the server will load the new web page.
- Whenever a user clicks on a link and as a result changes the current web page, all other shared browsers will be informed as well through a similar communication scheme.
- If the user's action causes a change of web page address in one or more "frames" that belong to a shared web page, JASBER will detect it and update the modified frame(s) in all other shared browsers as well.

The JASBER Applet can be located in a single and known web server. Once the HTML page from such server is loaded, JASBER starts up as an Applet and opens a communication channel to the appropriate JETS Server. Users who are interested in a shared browsing session could simply visit the known web server and launch the shared browser, hence joining the current collaborative session.

JASBER is a pure Java applet and, as such, it works in any platform with a standard Java Virtual Machine (JVM) implementation, such as Windows, Linux, Unix, Mac, etc.

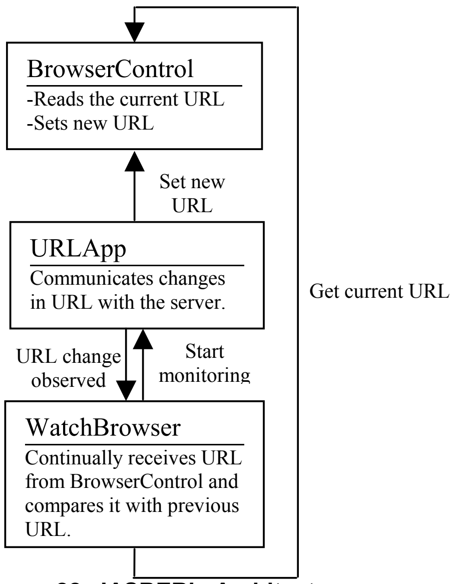


**Figure 22. JASBER's Architecture**

Standard Java classes such as Vector were used for the task of collecting the list of URLs gathered from the current browser page. The task of client-server communication is achieved through the use of JETS libraries, which provide a simple, yet effective, framework for JASBER. JASBER also uses some Netscape classes that provide means of negotiating security permissions with the user. These Netscape classes are, however, not compatible with browsers other than Netscape Communicator.

As it is shown in Figure 22, JASBER consists of three main modules. URLApp is a module that uses JETS classes to communicate with the server. WatchBrowser is a subclass of Java Thread and continually monitors the current URL of the shared browser. The module that directly reads and writes to the shared browser is BrowserControl. Since reading and changing the properties of a broswer requires permission of the users, this module makes use of Netscape security classes to meet such requirements.

## 6 JAVA AND VIRTUAL ENVIRONMENTS

An area where Java is perhaps not thought of as a viable option is in the development of distributed virtual environments. With the release of Java3D API however, developers can quickly and easily develop 3D applications and applets entirely in Java. We have developed such an application for the purpose of remote industrial training in a virtual setting. Two geographically distant users engage in a session where a trainer teaches the trainee how to install various hardware components. Java is used as the core technology, as well as for rendering, user interface, communication and multimedia integration. Figure 23 shows the architecture of the application and the various Java technologies used.

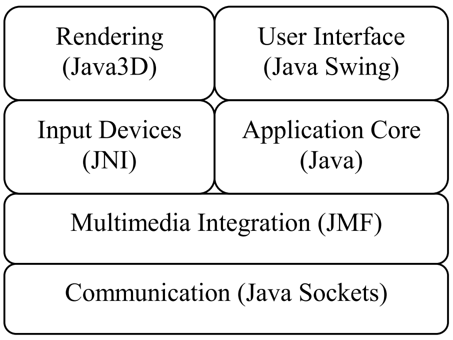

Rendering (Java3D)
User Interface (Java Swing)
Input Devices (JNI)
Application Core (Java)
Multimedia Integration (JMF)
Communication (Java Sockets)

**Figure 23. Java in Virtual Environment Architecture**

While Java3D provides a flexible and powerful 3D rendering API, the Java Native Interface has been used to connect our applications with several input devices such as CyberGlove, 6DoF miniBird and Phantom haptic device. Java Media Framework is used to integrate video and audio capabilities with the rest of the application.

All the Java technologies mentioned have allowed us to develop sophisticated distributed virtual environment applications involving immersive and stereoscopic displays, multiple input devices and mixed media environments (such as video inside 3D world). Figure 24 shows a training application using a CyberGlove to track the movements of the hand and a Phantom for haptic feedback (both connected via JNI), JMF transmission of trainer video and rendering it inside a Java3D scene.

The main advantage of using Java technology for the development of distributed virtual environments has been platform independence. This has been well illustrated during a recent move of our PC-based solutions to SGI-based ONYX machines. All our applications developed on Windows OS have been easily transferred to run on the Irix platform without a need for recompilation of the Java code. Hence we are now able to run our distributed virtual environment applications on both platforms. Figure 25 shows the same training application running on SGI ONYX.

In addition to inheriting the previously mentioned beneficial features of Java such as interoperability, the high-level programming interface of Java3D has the added benefit of allowing quick development of 3D applications without expertise in the field of computer graphics. The ability to include synthetic media (3D graphics) in Java applications and applets opens up yet another avenue in the field of multimedia collaboration where Java can have a significant impact.

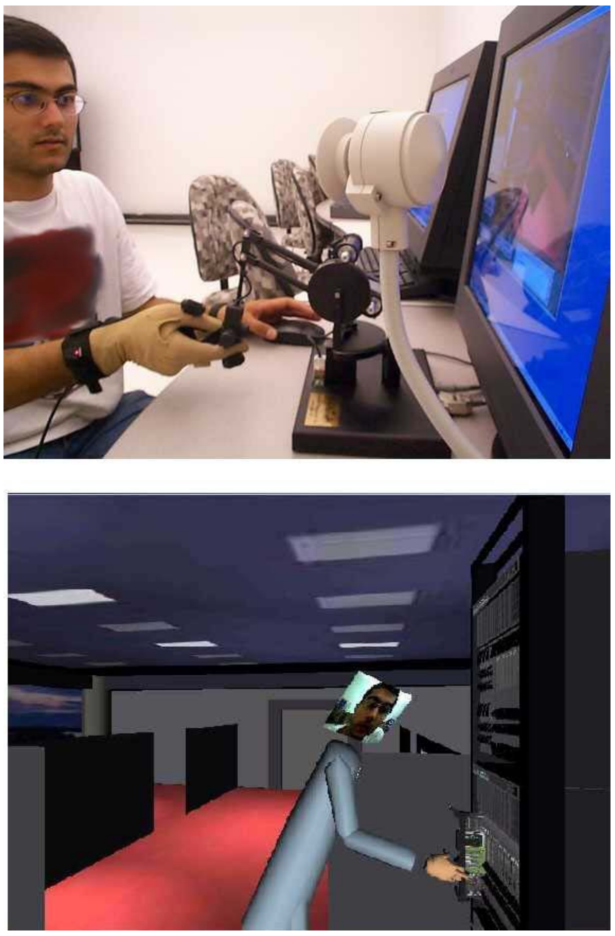


**Figure 24. Remote Training Using Multiple Input Devices, Video and 3D Media**

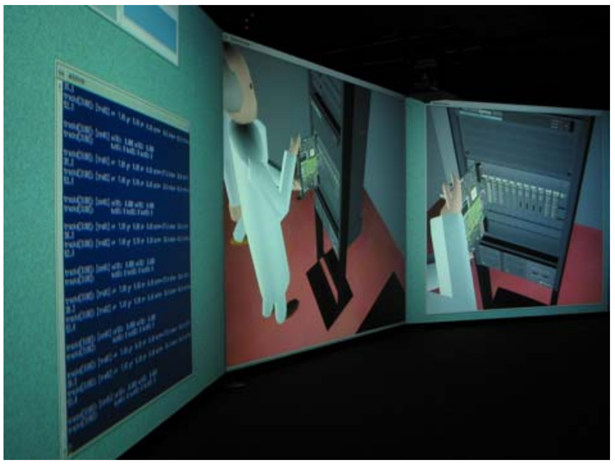


**Figure 25. Java3D on SGI ONYX**

## 7 RELATED WORKS

There are many Java-based collaboration systems, none of which offer a management or moderation feature similar to JETS or JASMINE. Kuhmünch [18] has developed a Java Remote Control Tool, which allows the control and synchronization of distributed Java applications and applets. Similar to JETS, this approach uses an API that Java applets and applications must use to become shareable by the system. The Java Shared Data Toolkit (JSDT) from JavaSoft is also an API-based framework. Habanero [16] is an approach that supports the development of collaborative environments. Habanero is in its terms a framework that helps developers create shared applications, either by developing a new one from scratch or by altering an existing single-user application which has to be modified to integrate the new collaborative functionality. Instead of using applets, which can be embedded in almost every browser, the Habanero system uses so-called “Happlets” which need a proprietary browser to be downloaded and installed on the client site. Java Collaborative Environment (JCE) has been developed at the National Institute of Standards and Technology (NIST) coming up with an extended version of the Java-AWT [20] called Collaborative AWT (C-AWT). In this approach AWT-components must be replaced by the corresponding C-AWT components.

All these approaches propose the use of an API, which has the cost of modifying the source-code of an application, re-implementing it or to design and implement a new application from scratch in order to make it collaborative. Another possible approach is the use of X Window System protocol like SharedX [17] or technologies based on NetMeeting [19] under Windows. However, this approach is not amenable to any group of users because NetMeeting runs only on Windows and SharedX needs the installation of an X Server and X Clients. Moreover NetMeeting does not provide the moderation capabilities presented in this work.

## 8 CONCLUSIONS

We have presented a number of Java based Multimedia Collaboration Tools. jStreaming stands by itself and allows streaming of H.263 video among Java enabled peers. JETS allows a group of users to collaborate in a shared whiteboard, where various medium are shared amongst the various participants of a Jets session. One such medium is H.263 streams, where jStreaming is used to allow the various users to share a streaming video. JASMINE goes one step further and allows a group of users to share almost any Java Applet or application, even those designed without collaboration in mind. JASMINE accomplishes collaboration without requiring any change in the Applets. Finally JASBER allows a group of users to share a web browsing session so that everyone can see the same content as everyone else. JASBER makes use of a JETS server to control the consistency among the various users. The COSMOS framework presents new possibilities in using Java for developing Collaborative Virtual Environments.

The set of applications presented show the great potential of the combination of Java and the Internet for Collaborative work.

jStreaming and JETS have been certified in Sun’s 100% Pure Java™ program as well as Novell’s “Yes” logo program. A former version of jStreaming has been the third prize winner in the ACM/IBM Quest for Java’97. A former version of JETS has been a First Prize Winner in the ACM/IBM Quest for Java’98. jStreaming and JETS have been licensed to several companies in three countries so far. jStreaming has also led to the 2001 OCRI Futures Award – Student Entrepreneur of the Year Award.

## ACKNOWLEDGEMENTS

We acknowledge the financial assistance of the Brazilian Ministry of Education Agency’s CAPES scholarship, the Ontario Research and Development Challenge Fund, the TeleLearning Network of Centres of Excellence, the Communication and Information Technology Ontario (CITO) and Newbridge Networks/Alcatel.

## REFERENCES

[1] J. C. de Oliveira; S. Shirmohammadi and N. D. Georganas, “Collaborative Virtual Environments Standards: A Performance Evaluation”, IEEE DiS-RT’99, Greenbelt, MD, October 1999.

[2] J. C. de Oliveira, X. Shen and N. D. Georganas Collaborative Virtual Environment for Industrial Training and e-Commerce, *Invited Paper*, IEEE VRTS'2000 (Globecom'2000 Conference's Workshop on Application of Virtual Reality Technologies for Future Telecommunication Systems), San Francisco, CA, November 2000.

[3] J. C. de Oliveira, M. Hosseini, S. Shirmohammadi, M. Cordea, E. Petriu, D. Petriu and N. D. Georganas – “VIRTUAL THEATER for Industrial Training: A Collaborative Virtual Environment”, Proc. 4th World Multi-Conference on Circuits, Systems, Communications & Computers (CSCC 2000), Vouliagmeni, Greece, July 2000.

[4] International Telecommunication Union, Telecommunication Standardization Sector - Audiovisual and Multimedia Systems – Infrastructure of audiovisual services – Coding of Moving Video – “Video Coding for low bitrate communication” – ITU-T Recommendation H.263, February 1998.

[5] S. McCane; V. Jacobson. - "vic: A Flexible Framework for Packet Video" – ACM Multimedia’95, San Francisco, CA, November 1995.

[6] S. Shirmohammadi, A. El Saddik, N. D. Georganas, and R. Steinmetz, "JASMINE: A Java Tool for Multimedia Collaboration on the Internet", Journal of Multimedia Tools and Applications Vol. 18, No.3, 2002

[7] S. Shirmohammadi and N. D. Georganas, JETS: Java-Enabled TeleCollaboration System", Proc. IEEE Multimedia Systems'97, Ottawa, June 1997.

[8] S. Shirmohammadi, J. C. Oliveira and N. D. Georganas, "Applet-Based Telecollaboration: A Network-centric Approach", IEEE Multimedia, Spring/Summer 1998.

[9] Multimedia Communications Research Laboratory, “JETS2000 User Manual”, 2001

[10] V. Darlagiannis and N. D. Georganas, “Virtual Collaboration and Media Sharing using COSMOS,” Proc. 4th WORLD MULTICONFERENCE on Circuits, Systems, Communications & Computers (CSCC 2000), Greece, July 2000

[11] S. Shirmohammadi, L. Ding, and N.D. Georganas, "An Approach for Recording Multimedia Collaborative Sessions: Design and Implementation", Journal of Multimedia Tools and Applications, Vol. 19, No.1, 2003

[12] http://www.javasoft.com

[13] http://jStreaming.com/~jauvane/H263Decoder/JDK1.1

[14] http://www.mcrlab.uottawa.ca/jets

[15] Multimedia Communication Forum Inc., “Multimedia Communication Quality of Service”, MMCF document MMCF/95-010, Approved Rev 1.0, September 24, 1995

[16] A. Chabert et al, “Java Object Sharing in Habanero”, Communications of the ACM, Volume 41, No. 6, June 1998, pp. 69-76.

[17] D. Garfinkel, B. Welti, T. Yip, “HP SharedX: A Tool for Real-Time Collaboration”, Hewlett-Packard Journal, April 1994, pp. 23-26.

[18] C. Kuhmünch, T. Fuhrmann, G. Schöppe, “Java Teachware - The Java Remote Control Tool and its Applications”, Proc. ED-MEDIA/ED-TELECOM'98, Freiburg, Germany, 1998.

[19] Microsoft Corporation, “NetMeeting Resource Kit”, <http://www.microsoft.com/windows/NetMeeting/Corp/reskit/>, 1999, last accessed: Oct. 18, 2000.

[20] H. Abdel-Wahab et al “An Internet Collaborative environment for Sharing Java Applications” IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems (FTDCS'97), October 29 - 31, 1997, pp. 112-117.